Statistical Techniques in Robotics (16-831, S21) Lecture #21 (Wednesday, April 21)

Policy Gradient Methods, Actor-Critic

Lecturer: Kris Kitani *Scribes: Andy Wei, Yi-Chun Chen*

1 Review

In the last lecture, we talked model-free value-based control, value function approximation, and an introduction to policy-based control. In this note, we will cover policy gradient method, a type of policy-based control method, and actor-critic, a hybrid of policy-based and value based control. In this section, we will review model-free control algorithm and value function approximation.

1.1 Model-Free Control

Model-free control aims to estimate the optimal policy without knowing the exact environment model; that is, the algorithm uses sampled value function to improve policy. Fig 1 shows the pipeline of model-free control. One can use any model-free prediction methods, e.g. TD Prediction, Monte-Carlo Prediction, when doing policy evaluation. We only listed the Monte-Carlo control in section 1.1.1 to let this note be concise.

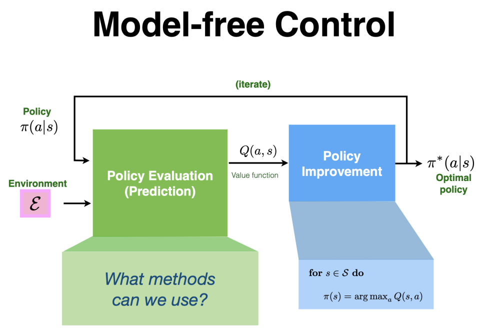

Figure 1: Model-Free Control Pipeline

1.1.1 Model-Free Control Methods Example: Monte-Carlo Control

The below listed the Monte-Carlo Control algorithm as Algorithm 1. Line 4 and line 5 are the monte-carlo prediction. Line 6 is the epsilon greedy policy, which forces the algorithm to explore random policy with probability ϵ.

Algorithm 1 MC Control($\pi_\epsilon, \epsilon, \alpha$)

```
1: for $e = 0, \cdots, E$ do
2:     $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \epsilon | \pi_\epsilon$
3:     for $t = 0, \cdots, T$ do
4:         $Q(a^{(t)}, s^{(t)}) \leftarrow Q(a^{(t)}, s^{(t)}) + \alpha[G^{(t)} - Q(a^{(t)}, s^{(t)})]$
5:         $a^* = \text{argmax}_a Q(a, s^{(t)})$
6:         $\pi_\epsilon(a|s^{(t)}) \leftarrow \frac{\epsilon}{\mathcal{A}(s^{(t)})} + \mathbf{1}[a = a^*](1-\epsilon)\ \forall a$
7:     end for
8: end for
```

1.2 Value Function Approximation

With finite state and action space, previously we only talked about a tabular value function. However, it is infeasible to use a tabular value function when the state or action space is infinite, as it may require infinite memory to store values for all combination of states and actions; this is where the value function approximation helps. The value function approximation aims to learn a function $V_\theta(s)$ that approximates the real value function $V(s)$ using some parameter θ, where s is the input state. Thus, the objectives will be finding a the optimal θ

$$\hat{\theta} = \text{argmin}_\theta \mathbb{E}_p[(V(s) - V_\theta(s))^2]$$

where p is the state visitation distribution. The solution can be found using stochastic gradient descent

$$\nabla_\theta L_\theta = -(V(s) - V_\theta(s)) \frac{\partial}{\partial \theta} V_\theta(s)$$

However, as we don't know the value function V(s), we use the estimate of the value function

$$V(s) = \mathbb{E}[G^{(t)}] \approx G^{(t)}$$

,with such estimation being done by model-free prediction methods we've covered, e.g. Monte-Carlo, TD. Similarly, the Q-value can be approximated using $Q_\theta(s, a)$. Such approximator can be linear function, neural networks, or even deep neural networks.

2 Summary

2.1 Policy Gradient Methods

Policy gradient methods is a type of reinforcement learning algorithm that directly learns the policy without estimating the value function. The goal of policy gradient methods is to find the policy parameters that maximizes the expected return

$$\begin{aligned} \hat{\theta} &= \arg\max_\theta \mathbb{E}_{p_\theta(\zeta)} \left[\sum_{t=0}^{T} r^{(t)} \right] \\ &= \arg\max_\theta J(\theta) \end{aligned}$$

, where

$$
\begin{aligned}
&p_\theta(\zeta) = p(s^{(0)})\Pi_{t=0}^{T}\pi_\theta(a^{(t)}|s^{(t)})p(s^{(t+1)}|s^{(t)}, a^{(t)}) \\
&\zeta = \{s^{(0)}, a^{(0)}, s^{(1)}, a^{(1)}, ..., s^{(T)}, a^{(T)}\} \\
&r^{(t)} \triangleq r(s^{(t+1)}, a^{(t)}, s^{(t)})
\end{aligned}
$$

2.1.1 Gradient Ascent

The way to find the policy that maximizes the expected return is *gradient ascent.* First, we make a linear approximation of the objective function with quadratic regularization.

$$
\hat{\theta} = \arg\max_{\theta} \{\alpha(J(\theta^{'}) + \langle\theta - \theta^{'}, \nabla_\theta^{'} J(\theta^{'})\rangle - \frac{1}{2}||\theta - \theta^{'}||^2)\}
$$

Then, we want to solve for the parameters that optimizes the Lagrangian, and we can get the parameters update equation.

$$
\begin{aligned}
&\nabla_\theta\{\alpha(J(\theta^{'}) + \langle\theta - \theta^{'}, \nabla_\theta^{'} J(\theta^{'})\rangle - \frac{1}{2}||\theta - \theta^{'}||^2)\} = 0 \\
\Rightarrow\ &\alpha\nabla_{\theta'} J(\theta^{'}) - \theta + \theta^{'} = 0 \\
\Rightarrow\ &\theta \leftarrow \theta^{'} + \alpha\nabla_{\theta'} J(\theta^{'})
\end{aligned}
$$

Recall that

$$
J(\theta) = \mathbb{E}_{p_\theta(\zeta)}\left[\sum_{t=0}^{T} r^{(t)}\right] = \int_\zeta p_\theta(\theta) r(\zeta) d\zeta, \qquad r(\zeta) = \sum_{t=0}^{T} r^{(t)}
$$

Then, we can start to compute the gradient.

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \int_\theta p_\theta(\theta) r(\zeta) d\zeta &&\text{, by definition} \\
&= \int_\theta \nabla_\theta p_\theta(\theta) r(\zeta) d\zeta &&\text{, linearity of gradient} \\
&= \int_\theta p_\theta \frac{\nabla_\theta p_\theta(\theta)}{p_\theta} r(\zeta) d\zeta &&\text{, multiply by one} \\
&= \int_\theta p_\theta \nabla_\theta \ln p_\theta(\theta) r(\zeta) d\zeta &&\text{, derivative of log}
\end{aligned}
$$

Next, we want to decompose the term $\nabla_\theta \ln p_\theta(\theta)$ and bring it back to the gradient $\nabla_\theta J(\theta)$.

$$\begin{aligned}
\nabla_\theta \ln p_\theta(\theta) &= \nabla_\theta \left[\ln p(s^{(0)}) + \sum_{t=1}^{T} \ln \pi_\theta(a^{(t)}|s^{(t)}) + \ln p(s^{(t+1)}|s^{(t)}, a^{(t)}) \right] \\
&= \nabla_\theta \left[\sum_{t=1}^{T} \ln \pi_\theta(a^{(t)}|s^{(t)}) \right] \qquad \text{, only policy is dependent on theta} \\
\nabla_\theta J(\theta) &= \int_\theta p_\theta \nabla_\theta \ln p_\theta(\theta) r(\zeta) d\zeta \\
&= \int_\theta p_\theta \nabla_\theta \left(\sum_{t=1}^{T} \ln \pi_\theta(a^{(t)}|s^{(t)}) \right) r(\zeta) d\zeta \\
&= \mathbb{E}_{p_\theta} \left[\left(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a^{(t)}|s^{(t)}) \right) \left(\sum_{t=1}^{T} r^{(t)} \right) \right]
\end{aligned}$$

We consider the term $\nabla_\theta \ln \pi_\theta(a^{(t)}|s^{(t)})$ eligibility vector. Intuitively, it is a weighted sensitivity of the policy to change in parameter. Since $\nabla_\theta \ln \pi_\theta(a^{(t)}|s^{(t)}) = \frac{\nabla_\theta \pi_\theta(a^{(t)}|s^{(t)})}{\pi_\theta(a^{(t)}|s^{(t)})}$, one can interpret that low probability actions have bigger gradient while high probability actions have smaller gradient. And we consider the other term $r(\zeta) = \sum_{t=1}^{T} r^{(t)}$ episodic return. The intuition is it is goodness of the current policy. Large reward (good trajectory) means bigger gradient while small reward (bad trajectory) means small gradient.

It is impossible to sum over all possible trajectories to compute the expectation, so we can use approximation, e.g. Monte-Carlo estimate. Therefore, we can obtain that

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{p_\theta} \left[\left(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a^{(t)}|s^{(t)}) \right) \left(\sum_{t=1}^{T} r^{(t)} \right) \right] \\
&= \frac{1}{N} \sum_{n=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a^{(n,t)}|s^{(n,t)}) \right) \left(\sum_{t=1}^{T} r^{(n,t)} \right)
\end{aligned}$$

2.1.2 Monte-Carlo Policy Gradient and Reinforce

Let's look at the algorithm of Monte-Carlo policy gradient.

Algorithm 2 MC-Policy-Gradient

```
1: for $e = 1, \cdots, E$ do
2:     $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \mathcal{E}|\pi_\theta$
3:     $G^{(0)} = \sum_{t=0}^{T} r^{(t)}$
4:     for $t = 0, \cdots, T$ do
5:         $\theta \leftarrow \theta + \alpha \left( \nabla_\theta \log \pi_\theta(a^{(t)}|s^{(t)}) \right) \left( G^{(0)} \right)$
6:     end for
7: end for
8: return $\pi_\theta$
```

A problem of the gradient is that Monte Carlo estimate has high variance. There are two ways to reduce variance of policy gradient. The first one is to enforce causality. The original gradient involves the reward of entire trajectory. However, the update to the policy at time t should only depend on future reward. Therefore, we can write the gradient as

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_{n=1}^{N}\left(\sum_{t=0}^{T}\nabla_\theta \ln \pi_\theta(a^{(n,t)}|s^{(n,t)})\right)\left(\sum_{t=1}^{T} r^{(n,t)}\right)\\ &\approx \frac{1}{N}\sum_{n=1}^{N}\left(\sum_{t=0}^{T}\nabla_\theta \ln \pi_\theta(a^{(n,t)}|s^{(n,t)})\right)\left(\sum_{t'=t}^{T} r^{(n,t')}\right)\end{aligned}$$

The other way is to remove gradient bias with a baseline offset. We can observe that large rewards results in large gradient variance, so we can adjust the gradient by subtracting some offset b, and the expected reward still stays the same.

$$\nabla_\theta J(\theta) = \int_\theta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\theta)[r(\zeta) - b]d\zeta$$

Below shows that offset will not change the expected gradient.

$$\begin{aligned}\nabla_\theta J(\theta) &= \int_\theta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\theta)[r(\zeta) - b]d\zeta\\ &= \int_\theta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\theta) r(\zeta) d\zeta - \int_\theta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\theta) b d\zeta\\ &= \int_\theta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\theta) r(\zeta) d\zeta - \int_\theta p_\theta(\zeta)\frac{\nabla_\theta p_\theta(\zeta)}{p_\theta(\zeta)} b d\zeta\\ &= \int_\theta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\theta) r(\zeta) d\zeta - b\nabla_\theta \int_\theta p_\theta(\zeta) d\zeta\\ &= \int_\theta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\theta) r(\zeta) d\zeta - b\nabla_\theta(1)\\ &= \int_\theta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\theta) r(\zeta) d\zeta - 0\end{aligned}$$

It is worth noting that in the episodic reinforce algorithm [2], the policy parameter updates is subtracting a baseline.

Algorithm 3 Episodic Reinforce($\pi_t heta, \alpha$)

```
1: for $e = 1, \cdots, E$ do
2:     $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \mathcal{E}|\pi_\theta$
3:     $G^{(0)} = \sum_{t=0}^{T} r^{(t)}$
4:     for $t = 0, \cdots, T$ do
5:         $\theta \leftarrow \theta + \alpha[G^{(0)} - b]\nabla_\theta \log \pi_\theta(a^{(t)}|s^{(t)})$
6:     end for
7: end for
8: return $\pi_\theta$
```

We can interpret REINFORCE with REward Increment = Non-negative Factor × Offset Reinforcement × Characteristic Eligibility.

$$\theta \leftarrow \theta + \overbrace{\underbrace{\alpha}_{\text{Non-negative factor}}\underbrace{[G^{(0)} - b]}_{\text{Offset Reinforcement}}\underbrace{\nabla_\theta \log \pi_\theta(a^{(t)}|s^{(t)})}_{\text{Characteristic Eligibility}}}^{\text{Reward Increment}}$$

To sum, adding baseline reduces the variance. As the variance is reduced, you don't need to sample as many trajectories to estimate the return. From the figure[1], you can see that reinforce with baseline converges faster.

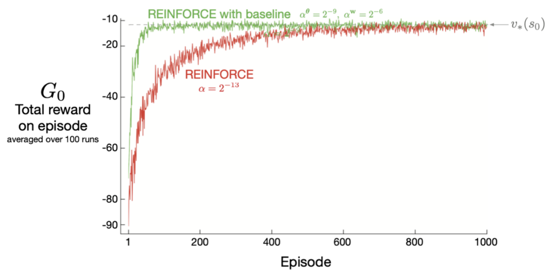

Figure 13.2: Adding a baseline to REINFORCE can make it learn much faster, as illustrated here on the short-corridor gridworld (Example 13.1). The step size used here for plain REINFORCE is that at which it performs best (to the nearest power of two; see Figure 13.1).

2.1.3 Policy Gradient

Putting the two modifications above together, we get the policy gradient algorithm.

Algorithm 4 Policy Gradient(π_θ, α, b)

```
1: for $e = 1, \cdots, E$ do
2:     $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \mathcal{E}|\pi_\theta$
3:     for $t = 0, \cdots, T$ do
4:         $G^{(t)} = \sum_{i=t}^{T} r^{(i)}$                                   (1) Enforce causality
5:         $\theta \leftarrow \theta + \alpha[G^{(t)} - b]\nabla_\theta \log \pi_\theta(a^{(t)}|s^{(t)})$   (2) Subtract baseline
6:     end for
7: end for
8: return $\pi_\theta$
```

Lastly, we list down the pros and cons of policy gradient methods.

Pros:

- Doesn't require model and learns from interaction
- Effective for high-dimensional or continuous action spaces (all methods so far assumed finite action space)

- Can encode prior knowledge when designing policy architecture
- Finds the optimal stochastic policy and naturally explores due to stochasticity

Cons:

- Gradient is typically high variance and leads to slow convergence
- Small step size leads to slow convergence
- Typically converge to local minima instead of global minima
- Exploding or zero gradient

2.2 Actor Critic

In the policy gradient, we have

$$\nabla_\theta J(\theta) = \mathbb{E}_{p_\theta}[(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a^{(t)}|s^{(t)}))(\sum_{t=1}^{T} r^{(t)})]$$

However, the $\sum_{t=1}^{T} r^{(t)}$ requires full-episode to compute. To prevent from such issues, Actor-Critic method uses model-free prediction for the return value estimator. For continuous state, we can further use function approximation for such estimate. Thus we have

$$\nabla_\theta J(\theta) = \mathbb{E}_{p_\theta}[(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a^{(t)}|s^{(t)}))Q_\phi(a^{(t)}, s^{(t)})]$$

where G_ϕ is function approximation of the return value parameterize by ϕ. Furthermore, the value function is included as baseline offset to remove the gradient bias of policy gradient, which is called advantage function.

$$A(a^{(t)}, s^{(t)}) = Q_\phi(a^{(t)}, s^{(t)}) - V_\psi(s^{(t)})$$

Thus the policy gradient with advantage function

$$\nabla_\theta J(\theta) = \mathbb{E}_{p_\theta}[(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(a^{(t)}|s^{(t)}))(\sum_{i=t}^{T} A_\phi(a, s))]$$

Below we review two advantage actor critic algorithm.

2.2.1 Advantage MC Actor-Critic

The Advantage-MC-Actor-Critic algorithm is listed as Algorithm 5. Line 5 is the function approximation update for value function. Line 6 is the function approximation update for return estimate. Line 7 iss the policy gradient.

Algorithm 5 Advantage-MC-Actor-Critic($\pi_\epsilon, Q_\phi, V_\psi, \alpha, \beta, \kappa$)

```
1: for $e = 0, \cdots, E$ do
2:     $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim\epsilon|\pi_\epsilon$
3:     for $t = 0, \cdots, T$ do
4:         $G^{(t)} \leftarrow \sum_{t=1}^{T} r^{(i)}$
5:         $\psi \leftarrow \psi + \kappa[G^{(t)} - V_\psi(s^{(t)})]\nabla_\psi V_\psi(s^{(t)})]$
6:         $\phi \leftarrow \phi + \beta[G^{(t)} - Q_\phi(a^{(t)}, s^{(t)})]\nabla Q_\phi(a^{(t)}, s^{(t)})$
7:         $\theta \leftarrow \theta + \alpha\nabla \log \pi_\theta(a^{(t)}|s^{(t)})\dot{Q}_\phi(a^{(t)}, s^{(t)})$
8:     end for
9: end for
```

2.2.2 Advantage TD Actor-Critic

Similarly, we can have Advantage TD Actor-Critic Algorithm, which listed as Algorithm 6

Algorithm 6 Advantage-TD-Actor-Critic($\pi_\epsilon, Q_\phi, V_\psi, \alpha, \beta, \kappa$)

```
1: for $e = 0, \cdots, E$ do
2:     $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim\epsilon|\pi_\epsilon$
3:     for $t = 0, \cdots, T$ do
4:         $\delta_V \leftarrow r^{(t)} + V_\psi(s^{(t+1)})$
5:         $\psi \leftarrow \psi + \kappa[\delta_V - V_\psi(s^{(t)})]\nabla_\psi V_\psi(s^{(t)})]$
6:         $\delta_Q \leftarrow r^{(t)} + Q_\phi(a^{t+1}, s^{(t+1)})$
7:         $\phi \leftarrow \phi + \beta[\delta_Q - Q_\phi(a^{(t)}, s^{(t)})]\nabla Q_\phi(a^{(t)}, s^{(t)})$
8:         $\theta \leftarrow \theta + \alpha\nabla \log \pi_\theta(a^{(t)}|s^{(t)})\dot{Q}_\phi(a^{(t)}, s^{(t)})$
9:     end for
10: end for
```

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[2] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.